Laporan Proyek Grafkom
(Racing in South Garda)

Kelompok 27
Louis Arthur Widya - C14210048 [Kelas B]
Ryan David Immanuel Tandean - C14210262 [Kelas B]
Adi Christian - C14210091 [Kelas A]

1. Pembagian Tugas

- Ryan David - C14210262 (Storyboard, Environment, Lighting + Shading, Skybox)
- Adi Christian - C14210091 (Storyboard, Kamera (TPP, FPP, dan free roam), Collision Detection)
- Louis Arthur - C14210048 (Kamera (TPP, FPP, dan free roam), Storyboard, Environment)

2. Rencana Story

Terdapat 2 pembalap rookie, pembalap mobil biru dan pembalap mobil merah. 2 pembalap ini merupakan rival yang mulai berkarir di waktu yang sama. Walaupun dibilang rival, pembalap mobil biru selalu kalah saat bertanding lawan pembalap mobil merah dengan jarak waktu sekitar 1-5 detik. Suatu ketika, *skill gap* diantara mereka makin menjauh. Pembalap mobil biru pun bertekad untuk mengalahkan pembalap mobil merah. Ia pun bersumpah akan mengalahkan pembalap mobil merah di pertemuan mereka selanjutnya, yaitu di arena balap South Garda. Ia pun berlatih keras menghafalkan track di arena South Garda. Hari pertandingan di South Garda pun tiba. Pembalap mobil merah memimpin dari awal pertandingan. Namun, hal ini berubah di lap terakhir. Pembalap mobil biru berhasil menyalip pembalap mobil merah dan memenangkan pertandingan. Ia pun direkrut oleh tim profesional karena berhasil memenangkan pertandingan tersebut.

3. Shading + Lighting

Lighting dan shading yang kami gunakan dalam project ini terdapat di setiap lampu jalanan yang ada dalam environment. Dengan menggunakan dir Light dan point Light Position. Directional Light berguna untuk memberikan efek seperti matahari bisa dibilang sumber cahayanya sedangkan untuk point Light Position digunakan untuk setiap masing masing tiang lampu agar dapat memberikan sebuah cahaya.

Sistem penerangan yang dipakai adalah lampu jalanan. Untuk membuatnya, pertama kami menentukan posisi x, y, z dari lampu jalanan. Lalu, koordinat - koordinat tersebut kami masukkan pada array _pointLightPositions agar nantinya muncul cahaya pada titik - titik tesebut. Untuk mengatur Shadingnya, kami mengganti-ganti angka pada diffuse dan specular.

Directional Light (direction, ambient, diffuse, specular)

```
// directional light
uniformsMap.setUniform( uniformName: "dirLight.direction", new Vector3f( x: -0.2f, y: -1.0f, z: -0.3f));
uniformsMap.setUniform( uniformName: "dirLight.ambient", new Vector3f( x: 0.05f, y: 0.05f, z: 0.05f));
uniformsMap.setUniform( uniformName: "dirLight.diffuse", new Vector3f( x: 0.4f, y: 0.4f, z: 0.4f));
uniformsMap.setUniform( uniformName: "dirLight.specular", new Vector3f( x: 0.5f, y: 0.5f, z: 0.5f));
```

Point Light Position (posisi masing masing tiang lampu)

```
Vector3f[] _pointLightPositions = {
        // ( 4.935E-1  4.140E-2 -4.916E-2)
        //( 4.764E-1  1.000E-3 -6.294E-2)
        new Vector3f( x: 0.4764f, y: 0.044f, z: -0.06294f),
        // ( 3.501E-1  4.420E-2  1.829E-1)
        //( 3.767E-1  1.000E-3  1.790E-1)
        new Vector3f( x: 0.3767f, y: 0.044f, z: 0.1790f),
        // ( 6.036E-2  4.420E-2  1.092E-1)
        //( 3.424E-1  1.000E-3  5.408E-1)
        new Vector3f( x: 0.3424f, y: 0.044f, z: 0.5408f),
        // (-2.591E-1  4.420E-2  1.106E-1)
        //( 4.353E-2  3.660E-2  1.151E-1)
        new Vector3f( x: 0.04353f, y: 0.044f, z: 0.1151f),
        // ( 3.251E-1  4.460E-2  5.189E-1)
        //(-2.313E-1  3.340E-2  1.070E-1)
        new Vector3f( x: -0.2313f, y: 0.044f, z: 0.107f)
};
```

Point Light Position (ambient, diffuse, specular)

```
for (int i=0;i<_pointLightPositions.length;i++){
    uniformsMap.setUniform( uniformName: "pointLights["+ i +"].position", _pointLightPositions[i]);
    uniformsMap.setUniform( uniformName: "pointLights["+ i +"].ambient", new Vector3f( x: 0.04f, y: 0.04f, z: 0.04f));
    uniformsMap.setUniform( uniformName: "pointLights["+ i +"].diffuse", new Vector3f( x: 0.5f, y: 0.5f, z: 0.5f)); // Adjusted diffuse value
    uniformsMap.setUniform( uniformName: "pointLights["+ i +"].specular", new Vector3f( x: 0.7f, y: 0.7f, z: 0.7f)); // Adjusted specular value
    uniformsMap.setUniform( uniformName: "pointLights["+ i +"].constant", value: 0.8f);
    uniformsMap.setUniform( uniformName: "pointLights["+ i +"].linear", value: 0.09f);
    uniformsMap.setUniform( uniformName: "pointLights["+ i +"].quadratic", value: 0.032f);
}
```

Pada Scene.frag juga jangan lupa untuk mengganti NR_POINT_LIGHTS sesuai jumlah pointLight yang diinginkan.

```
#define NR_POINT_LIGHTS 5
uniform PointLight pointLights[NR_POINT_LIGHTS];
```

Untuk mengganti warna pada lampu tambahkan uniformMap

```
    //lampu
    uniformsMap.setUniform( uniformName: "Lampu", new Vector3f( x: 1.0f, y: 1.0f, z: 1.0f));
```

Dan tambahkan juga di scene.frag

```
uniform vec3 Lampu;
```

```
    diffuse *= attenuation * Lampu;
    specular *= attenuation * Lampu;
```

4. Kamera

Kelompok kami membuat 3 macam kamera, yaitu free roam camera, third person camera, dan first person camera. Default POV saat program di run adalah POV free roam camera. Untuk berpindah-pindah POV berdasarkan kamera, bisa mengklik beberapa key.

- Key 1 untuk Free Roam Camera
- Key 2 untuk Third Person Camera Mobil Biru
- Key 3 untuk First Person Camera Mobil Biru

Pada free roam camera, mobil bergerak sesuai world's space.
Berikut beberapa key yang bisa digunakan Free Roam Camera:

- W → mobil gerak ke atas
- A → mobil gerak ke kiri
- S → mobil gerak ke bawah
- D → mobil gerak ke kanan
- LEFT → mobil rotate ke kiri
- RIGHT → mobil rotate ke kanan

Pada TPC dan FPC, mobil memiliki directionnya sendiri (local space).
Berikut beberapa key yang bisa digunakan di TPC dan FPC:

- W → mobil maju ke depan
- A → mobil rotate ke kiri
- S → mobil mundur ke belakang
- D → mobil rotate ke kanan

a) Free Roam Camera

Free roam camera berarti kita melihat dari tengah atas environment kita. Kami mengset posisi camera menggunakan `camera.setPosition(0.09f, 1.0f, 0.15f)` agar berada di tengah atas racing track kita.

Seperti yang sudah dijelaskan di atas, pada free roam cam, mobil bergerak sesuai world's space. Saat W dipencet, pada layar harus terlihat bahwa mobil bergerak ke atas. Padahal dalam POV mobil, hal itu berarti dia harus translate sebesar -Z. Hal serupa terjadi pada key A, S, D.

W → translateObject(0, 0, -Z)
A → translateObject(-X, 0, 0)
S → translateObject(0, 0, Z)
D → translateObject(X, 0, 0)

Key I, J, K, L, S, D berfungsi untuk menggerakkan kamera sesuai keinginan kita. Key LEFT dan RIGHT berfungsi merotate mobil ke kiri dan kanan. Caranya sama dengan yang diajarkan di kelas, yaitu dengan cara mentranslate mobil ke (0, 0, 0) lalu melakukan rotateObject sebesar (degree, 0, y, 0) untuk rotate kiri & (degree, 0, -y, 0) untuk rotate kanan lalu mentranslate kembali ke posisi semula.

b) Third Person Camera

Third person camera berarti kita fokus melihat objek dari sudut pandang orang ketiga. TPC kelompok kami fokus melihat objek mobil yang ada di depan (mobil biru) sehingga kami meletakkan TPC di belakang mobil biru. Caranya adalah dengan cara menyimpan center x, center y, center z dari mobil biru ke variabel x2, y2, z2. Lalu setPosition TPC ke x2, y2, z2 lalu mundurkan dan naikkan kamera menggunakan function moveBackwards (thirdPersonCamera.moveBackwards(0.02f)) dan function moveUp (thirdPersonCamera.moveUp(0.006f)).

Seperti yang sudah dijelaskan di atas, pada TPC, mobil bergerak sesuai local space (punya arah sendiri). Jadi ketika W dipencet, mobil & kamera akan maju dan ketika S dipencet, mobil & kamera akan mundur. Agar mobil mempunyai directionnya sendiri, kami membuat function sendiri pada Object.

```
Vector3f dir = new Vector3f(0, 0, 1);
```

```
public void moveForward(Float amount) {
    Matrix4f translationMatrix = new Matrix4f().translate(dir.x *
amount, dir.y * amount, dir.z * amount);
   model = model.mul(translationMatrix);
}

public void moveBackward(Float amount) {
    Matrix4f translationMatrix = new Matrix4f().translate(-dir.x *
amount, -dir.y * amount, -dir.z * amount);
   model = model.mul(translationMatrix);
}
```

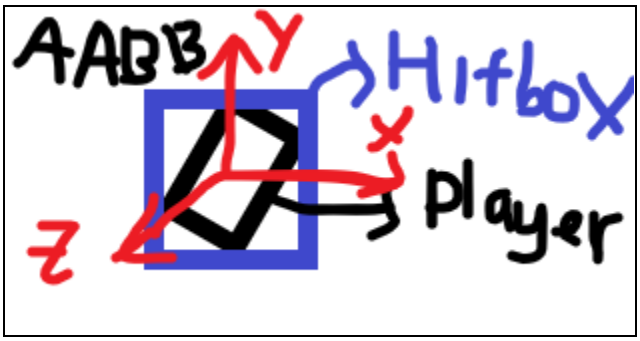
Agar objek maju, kami menggunakan objects.get(i).moveForward. Agar objek mundur, kami menggunakan objects.get(i).moveBackward. Agar kamera gerak saat objek gerak, kami letakkan kamera di belakang mobil dengan cara yang sudah dijelaskan di atas (setPosition TPC ke x2, y2, z2 lalu moveBackwards dan moveUp).

Key A dan D berfungsi untuk rotate mobil beserta kamera ke kiri dan kanan. Hal ini dilakukan dengan cara setPosition TPC ke x2, y2, z2 lalu setRotation sebesar (0, Math.toRadians(1.5f)*count+ Math.toRadians(90)) Count merupakan integer yang default valuenya 0. Saat A dipencet, count berkurang sebanyak 1. Saat D dipencet, count bertambah sebanyak 1. Count ini berfungsi untuk tahu arah rotate kameranya. Lalu perlu ditambah Math.toRadians(90) karena saat inisialisasi TPC, kami addRotation sebesar (0, Math.toRadians(90)).

c) First Person Camera
   First person camera berarti kita fokus melihat objek dari sudut pandang orang pertama. FPC kelompok kami fokus pada POV pengendara mobil biru sehingga kami meletakkan FPC di titik pusat mobil birunya. Caranya sama dengan TPC, namun tidak perlu menggunakan fungsi moveBackwards. Semua function di FPC sama dengan TPC, hanya ada perbedaan posisi kamera dan tidak perlu moveBackwards.

5. Collision Detection

Kelompok kami memakai konsep AABB (Axis-Aligned Bounding Box) Collision Detection, penggambarannya seperti ilustrasi berikut



“Player” Ibaratnya objek yang hendak kita buat memiliki hitbox. Pertama-tama, kita cek semua vertex dari Player. Lalu, kita dapat nilai min dan max untuk x, y, dan z nya. Nilai-nilai itu akan membuat semacam Hitbox 3D mengelilingi player. Pengecekan itu ada di cuplikan code berikut.

```
// Hitbox
public void calculateBoundingBox() {
    Vector3f min = new Vector3f(Float.POSITIVE_INFINITY);
    Vector3f max = new Vector3f(Float.NEGATIVE_INFINITY);

    // Check each vertice
    for (Vector3f vertex : vertices) {
        Vector4f transformedVertex = new Vector4f(vertex, 1.0f);
        model.transform(transformedVertex);

        // Update minimum and maximum values for each coordinate
        min.x = Math.min(min.x, transformedVertex.x);
        min.y = Math.min(min.y, transformedVertex.y);
        min.z = Math.min(min.z, transformedVertex.z);
        max.x = Math.max(max.x, transformedVertex.x);
        max.y = Math.max(max.y, transformedVertex.y);
        max.z = Math.max(max.z, transformedVertex.z);
    }

    if (boundingBox == null) {
        boundingBox = new AABB(min, max);
    } else {
        boundingBox.updateAABBMixMax(min, max);
    }
}
```

Setelah itu, nanti akan dilakukan checking pada semua object, apakah player collide dengan object lain. Jika player collide dengan object lain, maka transformasi yang dilakukan akan dibatalkan dan player akan kembali ke posisi sebelum transform. Checking apakah terjadi collide dengan object lain atau tidak

ada pada cuplikan code berikut.

```
public boolean AABBIntersects(AABB aabb1, AABB aabb2) {
    return !(aabb1.max.x < aabb2.min.x || aabb1.min.x > aabb2.max.x ||
             aabb1.max.y < aabb2.min.y || aabb1.min.y > aabb2.max.y ||
             aabb1.max.z < aabb2.min.z || aabb1.min.z > aabb2.max.z);
}
```

Sedangkan checking untuk membatalkan transformasi ada di code dari function transformasi yang bersangkutan. Konsepnya mirip-mirip, maka akan saya bahas sekali saja.

```
public void moveForwardCheckCollision(float amount, List<AABB> wallAABBs)
{
    Matrix4f originalModel = new Matrix4f(model); // Create a copy of the
original model matrix
    AABB originalBoundingBox = new AABB(boundingBox.getMin(),
boundingBox.getMax());

    // Calculate the new position after movement
    Matrix4f translationMatrix = new Matrix4f().translate(dir.x * amount,
dir.y * amount, dir.z * amount);
    model = model.mul(translationMatrix);
    calculateBoundingBox();

    // Check for collision with walls
    boolean collisionDetected = false;
    for (AABB wallAABB : wallAABBs) {
        if (boundingBox.AABBIntersects(boundingBox, wallAABB)) {
            // Collision detected, cancel movement
            model = originalModel; // Reset the model matrix to its
original state
            boundingBox = originalBoundingBox;
            collisionDetected = true;
            break;
        }
    }

    if (!collisionDetected) {
        // No collision detected, update player position
        for (Object child : childObject) {
            ((Sphere)child).moveForwardCheckCollision(amount, wallAABBs);
        }
    }
}
```

Awalnya, kita simpan nilai original, yaitu nilai sebelum transformasi.
Setelah itu, kita hitung transformasinya dengan cara merubah model sesuai transformasi yang berkaitan. Di cuplikan code ini, model diubah agar object bisa bergerak searah sumbu dir nya.

Sehabis pergerakan, dilakukan calculateBoundingBox() lagi untuk menghitung nilai hitbox min dan max.
Setelah itu, dilakukan pengecekan untuk tiap object lain, disini yaitu walls.

Jika ada collision, maka kembalikan nilai model dan boundingBox ke nilai original, yaitu nilai sebelum transformasi.

Jika tidak ada collision, maka lakukan function transformasi yang sama untuk semua child object juga.

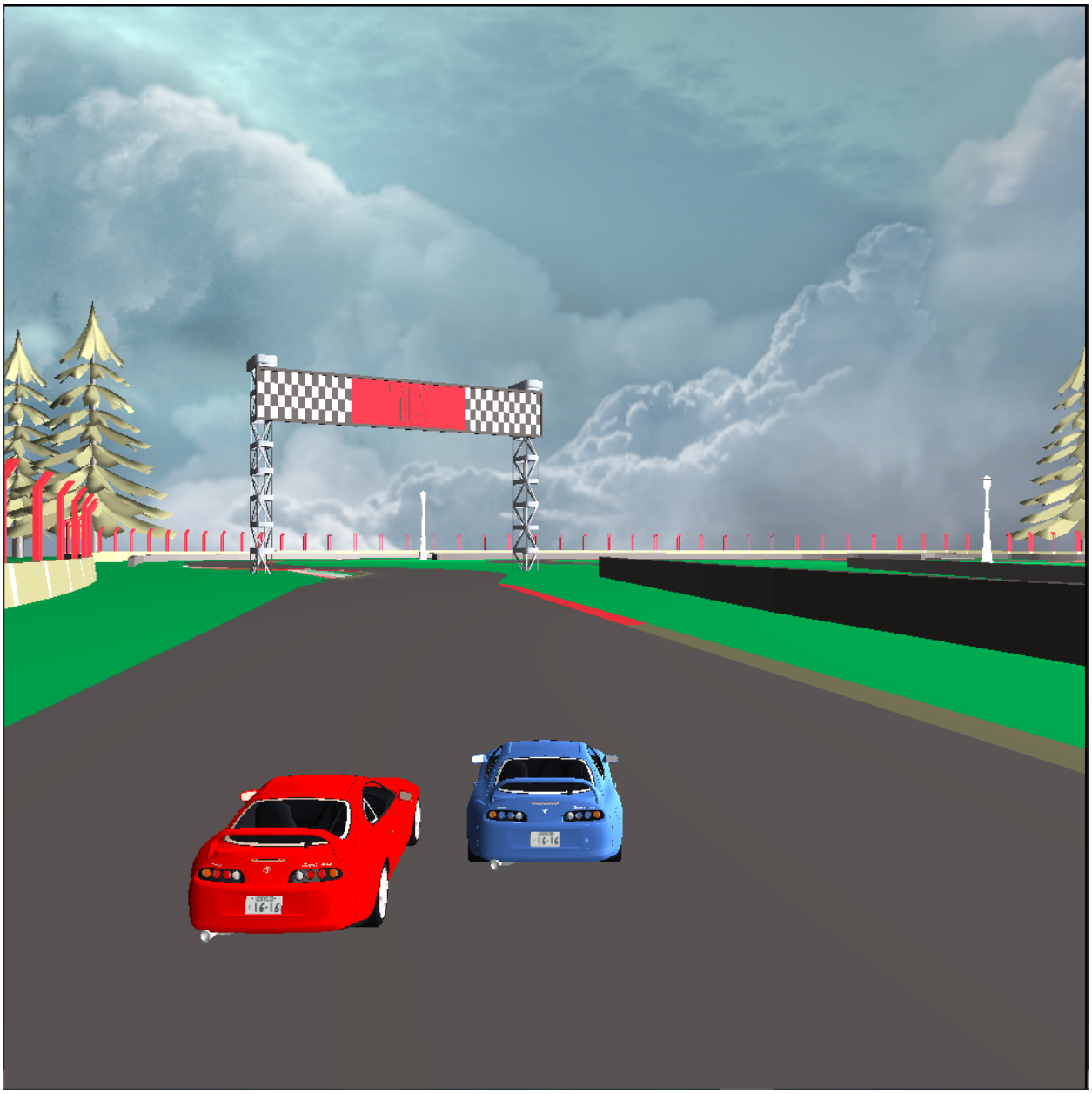
6. Optimization

Kami melakukan sedikit optimization dengan cara Culling face/sisi yang tidak menghadap ke kamera. Dengan begini, face/sisi yang tidak menghadap ke kamera tidak akan dirender sehingga bisa menghemat resource.

```
// Cull
GL30.glCullFace(GL30.GL_CULL_FACE);
GL30.glCullFace(GL_BACK);
```

7. Skybox

Kami membuat skybox ini dengan menggunakan konsep Cube dimana setiap sisi dari cube diberi sebuah texture dengan format .png.



Gambar awan tersebut merupakan Skybox yang dimaksud